# Reya Network Security Review

## Pashov Audit Group

Conducted by: Peakbolt, 0xbepresent, pontifex

April 30th 2024 - May 3rd 2024

# Contents

# 1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

# 2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

# 3. Introduction

A time-boxed security review of the **reya-network** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

# 4. About Reya Network

Reya Network is a trading-optimised modular L2 for perpetuals. The chain layer is powered by Arbitrum Orbit and is gas-free, with transactions ordered on a FIFO basis. The protocol layer directly tackles the vertical integration of DeFi applications by breaking the chain into modular components to support trading, such as PnL settlements, margin requirements, liquidations.

# 5. Risk Classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## 5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## 5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

## 5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

# 6. Security Assessment Summary

*review commit hash* - 5ef6ed67b5478b734485f04ebc6167630812092c

*fixes review commit hash* - 95671a3dd756d33a8cdce40dd728e388e23d2bce

## Scope

The following smart contracts were in scope of the audit:

- `TransferCollateral`
- `ExecutionModule`
- `AccountCollateral`
- `BackstopLPConfiguration`
- `CollateralConfiguration`
- `CollateralPool`
- `Market`
- `ConfigurationModule`
- `PassivePerpInstrumentModule`
- `ExecutionModule`
- `OrderModule`
- `Configuration`
- `PrbMathHelper`
- `Timer`
- `Events`
- `Errors`
- `DataTypes`

# 7. Executive Summary

Over the course of the security review, Peakbolt, 0xbepresent, pontifex engaged with Reya Labs to review Reya Network. In this period of time a total of **9** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Reya Network |
| **Repository** | https://github.com/Reya-Labs/reya-network |
| **Date** | April 30th 2024 - May 3rd 2024 |
| **Protocol Type** | Perpetuals Trading L2 |

## Findings Count

| **Severity** | **Amount** |
|---|---|
| High | 2 |
| Medium | 4 |
| Low | 3 |
| **Total Findings** | **9** |

# Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [H-01] | Lack of withdrawal limits check | High | Resolved |
| [H-02] | Ignoring withdrawal limits during accounts merging | High | Resolved |
| [M-01] | LPs can withdraw immediately even with withdrawal cooldown | Medium | Resolved |
| [M-02] | New MarketConfigurationData causes existing orders to fail | Medium | Acknowledged |
| [M-03] | Bypassing collateral cap check | Medium | Resolved |
| [M-04] | matchOrders affected if withdrawals exceed the global limit | Medium | Resolved |
| [L-01] | Missing validation of non-zero value | Low | Resolved |
| [L-02] | Command execution could fail | Low | Acknowledged |
| [L-03] | Missing maxExposureFactor in Errors.ExceededMaxExposure | Low | Resolved |

# 8. Findings

## 8.1. High Findings

## [H-01] Lack of withdrawal limits check

### Severity

**Impact:** Medium

**Likelihood:** High

### Description

The provided update fully excludes withdrawal limits checks from the `CollateralConfiguration.checkWithdrawLimits` function and puts them into the `AccountCollateral.updateBalance` function. So `AccountCollateral.transferFunds` and `ExposedModule.updateCollateralShares` functions have neither `GlobalCollateralConfiguration.checkWithdrawLimits` nor `CollateralConfiguration.checkWithdrawLimits` check since they bypass the `updateBalance` function.

```
function transferFunds
      (uint128 fromAccountId, uint128 toAccountId) internal {
        if (fromAccountId == toAccountId) {
            return;
        }

        Data storage fromAccountCollateral = load(fromAccountId);
        address[] storage collaterals = GlobalCollateralConfiguration.load
          ().collaterals;
        for (uint256 i = 0; i < collaterals.length; i++) {
            address activeCollateral = collaterals[i];
            int256 amount = fromAccountCollateral
                .collateralShares[activeCollateral];

            if (amount == 0) {
                continue;
            }

>>          updateShares(fromAccountId, activeCollateral, -amount);
            updateShares(toAccountId, activeCollateral, amount);
        }
    }
```

```
// Export some internal functions
contract ExposedModule {
    using CollateralPool for CollateralPool.Data;
<...>
    function updateCollateralShares
      (uint128 cpId, address collateral, int256 sharesDelta) external {
>>      CollateralPool.updateCollateralShares(CollateralPool.exists
  (cpId), collateral, sharesDelta);
    }
}
```

## Recommendations

Consider implementing corresponding checks for these branches.

# [H-02] Ignoring withdrawal limits during accounts merging

## Severity

**Impact:** High

**Likelihood:** Medium

## Description

The `CollateralPool.merge` merges collateral pool `child` into the collateral pool `parent` ignoring withdrawal limits of the `child` pool.

```
function merge(Data storage parent, Data storage child) internal {
<...>
            // transfer funds from the child collateral pool to the parent
            {


                for (uint256 i = 0; i < collaterals.length; i++) {
                    address collateral = collaterals[i];
                    int256 amount = child.collateralShares[collateral].toInt();

                    if (amount == 0) {
                        continue;
                    }

                    CollateralConfiguration.exists(parentId, collateral);
>>                  _updateCollateralShares(child, collateral, -amount);
                    _updateCollateralShares(parent, collateral, amount);
                }
            }
```

Funds can be withdrawn directly after `merge` since `child` pool withdrawal limits are not transferred to the `parent` pool.

# Recommendations

Consider transferring all withdrawal limits from the `child` pool to the `parent` pool.

# 8.2. Medium Findings

# [M-01] LPs can withdraw immediately even with withdrawal cooldown

## Severity

**Impact:** Medium

**Likelihood:** Medium

## Description

The withdrawal cooldown period for Backstop LPs is now configured using `backstopLPConfig.withdrawCooldownDurationInSeconds`, such that the withdrawal cooldown period is turned on when it is greater than 0 and turned off when it is equal 0.

However, the issue is that Backstop LPs can start the withdrawal timer before the withdrawal cooldown is turned on, allowing them to withdraw immediately even after the withdrawal cooldown is enabled.

1. Suppose the cooldown period for withdrawal has been disabled for a while, so `withdrawCooldownDurationInSeconds == 0`.
2. Knowing that the cooldown period will be enabled soon, Backstop LPs proceed to start the withdrawal timer using `announceBackstopLpWithdraw()`.
3. Protocol now enables withdrawal cooldown to `x` seconds.
4. However, those Backstop LPs who had announced withdrawal earlier will be able to withdraw immediately as the withdrawal period is active.

```
function announceBackstopLpWithdraw(Account.Data storage account) internal {
        CollateralPool.Data storage collateralPool = AccountCollateral.getPool
          (account.id);

        uint128 backstopLpAccountId = backstopLPConfig.accountId;

        if (backstopLpAccountId != account.id) {
            revert Errors.AccountIsNotBackstopLp(
                account.id,
                backstopLpAccountId,
                block.timestamp
            );
        }

        Timer.Data storage backstopLpWithdrawTimer = Timer.loadOrCreate(
            backstopLpTimerId(backstopLpAccountId)
        );
        if (block.timestamp < backstopLpWithdrawTimer.startTimestamp) {
            revert Errors.BackstopLpCooldownPeriodAlreadyActive(
                backstopLpAccountId,
                backstopLpWithdrawTimer.startTimestamp,
                block.timestamp
            );
        }
        if (backstopLpWithdrawTimer.isActive()) {
            revert Errors.BackstopLpWithdrawPeriodAlreadyActive(
                backstopLpAccountId,
                block.timestamp
            );
        }

        backstopLpWithdrawTimer.schedule(
            block.timestamp
            +
            backstopLPConfig.withdrawCooldownDurationInSeconds,
            backstopLPConfig.withdrawDurationInSeconds
        );
    }
```

# Recommendations

Prevent Backstop LPs from starting withdrawal timer using `announceBackstopLpWithdraw()` by reverting when `backstopLPConfig.withdrawCooldownDurationInSeconds == 0`.

# [M-02] New MarketConfigurationData causes existing orders to fail

# Severity

**Impact:** Medium

**Likelihood:** Medium

# Description

There are three new storage variables (`depthFactor`, `maxExposureFactor`, `maxPSlippage`) in `MarketConfigurationData` that are used to perform slippage and exposure checks in `Market.getPSlippage()`.

The issue that that these new storage variables will be initialized to zero for the existing `Market` as they were previously un-used before this upgrade.

When `maxExposureFactor == 0`, it will cause `getPSlippage()` to incorrectly revert even when net exposure is valid.

```
function getPSlippage(
        Data storage self,
        SD59x18 deltaBase,
        UD60x18 oraclePrice
    )
        internal
        view
        returns (SD59x18 pSlippage)
    {
        MarketConfigurationData memory marketConfig = getConfig(self);

        uint256 riskMatrixIndex = marketConfig.riskMatrixIndex;
        UD60x18 depthFactor = marketConfig.depthFactor;
        UD60x18 maxExposureFactor = marketConfig.maxExposureFactor;
        UD60x18 maxPSlippage = marketConfig.maxPSlippage;

        (
          UD60x18maxExposureShort,
          UD60x18maxExposureLong,
          SD59x18[]memoryexposures
        ) = getPoolMaxExposures(self
        SD59x18 deltaExposure = convertBaseToExposure(deltaBase, oraclePrice);

        SD59x18 netExposure = exposures[riskMatrixIndex].add(deltaExposure);
        UD60x18 maxExposure = netExposure.lt
          (ZERO_sd) ? maxExposureShort : maxExposureLong;

        //@audit when maxExposureFactor == 0, this will revert for valid net
        // exposure too
        if (netExposure.abs().intoUD60x18().gte(maxExposure.mul
          (maxExposureFactor))) {
            revert Errors.ExceededMaxExposure(netExposure, maxExposure);
        }

        pSlippage = computePSlippage(
          {netExposure:netExposure,
          maxExposure:maxExposure,
          depthFactor:depthFactor}
        );

        if (pSlippage.abs().intoUD60x18().gt(maxPSlippage)) {
            revert Errors.ExceededPSlippage(pSlippage, maxPSlippage);
        }
    }
```

# Recommendations

Ensure that the new storage variables are initialized to default values when performing the contract upgrade. Otherwise, handle the uninitialized value in `getPSlippage()`.

# [M-03] Bypassing collateral cap check

## Severity

**Impact:** High

**Likelihood:** Low

## Description

The `CollateralPool.merge` merges collateral pool `child` into collateral pool `parent` bypassing cap check in `updateCollateralShares` function since it directly invokes `_updateCollateralShares` function.

```
function merge(Data storage parent, Data storage child) internal {
<...>
            // transfer funds from the child collateral pool to the parent
            {


                for (uint256 i = 0; i < collaterals.length; i++) {
                    address collateral = collaterals[i];
                    int256 amount = child.collateralShares[collateral].toInt();

                    if (amount == 0) {
                        continue;
                    }

                    CollateralConfiguration.exists(parentId, collateral);
                    _updateCollateralShares(child, collateral, -amount);
>>                  _updateCollateralShares(parent, collateral, amount);
                }
            }
```

In case the `parent` pool balance before the merge is not empty the `collateralConfig.baseConfig.cap` can be exceeded.

## Recommendations

Consider checking the collateral cap in the `merge` function.

# [M-04] `matchOrders` affected if withdrawals exceed the global limit

## Severity

**Impact:** High

**Likelihood:** Low

## Description

Within the `AccountCollateral::updateBalance` function, a validation was added to check that withdrawals do not exceed a global percentage for each X period of time (code line 120):

```
File: AccountCollateral.sol
107:     function updateBalance
  (Account.Data storage account, address collateral, int256 assets) internal {
108:         // Convert assets to shares
109:         int256 shares = GlobalCollateralConfiguration.convertToShares
  (collateral, assets);
110:
111:         // check withdrawal limits, globally and per collateral pool
112:         if (assets < 0) {
113:             uint256 withdrawnAssets = (-assets).toUint();
114:
115:             if (hasPool(account.id)) {
116:                 uint128 collateralPoolId = getPool(account.id).id;
117:                 CollateralConfiguration.exists
  (collateralPoolId, collateral).checkWithdrawLimits(withdrawnAssets);
118:             }
119:
120:             GlobalCollateralConfiguration.checkWithdrawLimits
  (collateral, withdrawnAssets);
121:         }
122:
123:         updateShares(account.id, collateral, shares);
124:     }
```

The issue arises when a fee amount is deducted from the account in `MatchOrderModule::executeMatchOrder`, specifically in line 85:

```
File: MatchOrderModule.sol
    function executeMatchOrder(
        address caller,
        uint128 marketId,
        uint128 accountId,
        uint128 exchangeId,
        uint128[] memory counterpartyAccountIds,
        bytes memory orderInputs
    )
        external
        override
        returns (bytes memory output)
    {
...
...
        MatchOrderFees memory matchOrderFees;
        (output, matchOrderFees) = market.executeMatchOrder
  ({ matchOrderInputs: matchOrderInputs });
        validateMatchOrderFees(matchOrderFees, creditExchangeFees);

        // deduct fees from the main account and track the total amounts of
// fees for protocol and exchange
        AccountCollateral.updateBalance(account, market.quoteCollateral, -
  (matchOrderFees.takerFeeDebit).toInt());
...
...
    }
```

The issue is that the transaction can be reverted in edge cases where the global limit is reached due to other withdrawals affecting the execution of match orders and also affecting the execution of commands coming from the new function added in `periphery/src/modules/ExecutionModule::execute` since they would have to generate a signature again if the `deadline` is not sufficient until `checkWithdrawLimits` allows withdrawals again.

# Recommendations

It is recommended that if subtraction from the account for fees occurs, then transactions should not be reversed. Otherwise, the execution of match orders will be affected in very specific cases.

## 8.3. Low Findings

## [L-01] Missing validation of non-zero value

Within `ConfigurationModule::setMarketConfiguration`, there is no validation that `MarketConfigurationData.maxSlippage` is not zero:

```
File: ConfigurationModule.sol
    function setMarketConfiguration(uint128 marketId, MarketConfigurationData memory config) external override {
        if (config.oracleNodeId == 0) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "ORCLN");
        }

        if (config.baseSpacing.eq(ZERO_ud)) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "BSSP");
        }

        NodeOutput.Data memory node =
            INodeModule(GlobalConfiguration.getOracleManagerAddress()).process(config.oracleNodeId);
        UD60x18 oraclePrice = UD60x18.wrap(node.price);

        if (config.priceSpacing.eq(ZERO_ud) || oraclePrice.lte(config.priceSpacing.mul(ud(1000e18)))) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "PRCSP");
        }

        if (!config.minimumOrderBase.mod(config.baseSpacing).eq(ZERO_ud)) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "MNOB");
        }

        // TODO: it should be less or equal than 0.01 but it breaks a lot of // testing doing so
        if (config.velocityMultiplier.gt(ud(1e18))) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "VLCTM");
        }

        if (config.depthFactor.eq(ZERO_ud)) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "DPTHF");
        }

        if (config.maxExposureFactor.gt(ONE_ud)) {
            revert Errors.InvalidMarketConfiguration(marketId, config, "MXEXF");
        }

        Market.Data storage market = Market.exists(marketId);
        market.onlyAuthorized(Permissions.PASSIVE_PERP_MARKET_CONFIGURATOR);

        MarketConfiguration.set(marketId, config);
    }
```

This could affect the calculation of `pSlippage` within the `Market::getPSlippage` function, as any non-zero value of `pSlippage` would cause the function to revert with an `ExceededPSlippage` error if `maxPSlippage` is zero (lines 301-303).

```
File: Market.sol

    function getPSlippage(
        Data storage self,
        SD59x18 deltaBase,
        UD60x18 oraclePrice
    )
        internal
        view
        returns (SD59x18 pSlippage)
    {
        MarketConfigurationData memory marketConfig = getConfig(self);

        uint256 riskMatrixIndex = marketConfig.riskMatrixIndex;
        UD60x18 depthFactor = marketConfig.depthFactor;
        UD60x18 maxExposureFactor = marketConfig.maxExposureFactor;
        UD60x18 maxPSlippage = marketConfig.maxPSlippage;

        (
  UD60x18maxExposureShort,
  UD60x18maxExposureLong,
  SD59x18[]memoryexposures
) = getPoolMaxExposures(self
        SD59x18 deltaExposure = convertBaseToExposure
  (deltaBase, oraclePrice);

        SD59x18 netExposure = exposures[riskMatrixIndex].add
  (deltaExposure);
        UD60x18 maxExposure = netExposure.lt
  (ZERO_sd) ? maxExposureShort : maxExposureLong;

        if (netExposure.abs().intoUD60x18().gte(maxExposure.mul
  (maxExposureFactor))) {
            revert Errors.ExceededMaxExposure(netExposure, maxExposure);
        }

        pSlippage = computePSlippage(
  {netExposure:netExposure,
  maxExposure:maxExposure,
  depthFactor:depthFactor}
);

        if (pSlippage.abs().intoUD60x18().gt(maxPSlippage)) {
            revert Errors.ExceededPSlippage(pSlippage, maxPSlippage);
        }
    }
```

It is advisable to evaluate that `maxSlippage` is not zero within the `ConfigurationModule::setMarketConfiguration` function.

# [L-02] Command execution could fail

The `CommandType` enum was modified with the removal of the `PropagateCashflow` element, which was previously assigned to the value `4`. The change will cause the value of `TransferBetweenMarginAccounts` to be changed from `5` to `4`.

That could cause issues for commands constructed right before the contract upgrade and then executed after the upgrade. If those commands contain `TransferBetweenMarginAccounts`, it would be constructed based on the old value `5` before the upgrade. When they are executed after the upgrade, these commands will fail as they do not match the contract implementation.

```
enum CommandType {
    Deposit, // (core command) deposit collaterals
    Withdraw, // (core command) withdraw collaterals
    DutchLiquidation, // (core command) dutch liquidation of an account
    MatchOrder, // (market command) propagation of matched orders
    //@audit PropagateCashflow is removed from enum, causing
    // TransferBetweenMarginAccounts to change from 5 to 4
    //PropagateCashflow, // (market command) propagation of realized PnL
    TransferBetweenMarginAccounts //
    //(core command) transfer between two margin accounts

}
```

## [L-03] Missing `maxExposureFactor` in `Errors.ExceededMaxExposure`

A `maxExposureFactor` was added to adjust the max exposure during the slippage check. However, the `maxExposureFactor` value is not present in the custom error `Errors.ExceededMaxExposure`, which will not emit the correct parameters when the transaction reverts.

To resolve this, add `maxExposureFactor` to `Errors.ExceededMaxExposure`.

```
function getPSlippage(
        Data storage self,
        SD59x18 deltaBase,
        UD60x18 oraclePrice
    )
        internal
        view
        returns (SD59x18 pSlippage)
    {
        ...
        if (netExposure.abs().intoUD60x18().gte(maxExposure.mul
          (maxExposureFactor))) {
           //@audit error does not contain the maxExposureFactor value
            revert Errors.ExceededMaxExposure(netExposure, maxExposure);
        }

        ...
    }
```